Maximizing Benefits of Adaptive Test Techniques by Integration of ML and Test Infrastructure Improvements

Abhijit Sathaye
Principal Engineer
Manufacturing and Product Engineering
Intel Corporation
Hillsboro, OR, USA
abhijit.sathaye@intel.com

Abstract:

Testing Si devices in high volume manufacturing (HVM) is getting more challenging and expensive, owing to the increasing complexity of the devices under test. Semiconductor companies are adding more capabilities on a package, transistor density is going up, end user applications are more varied than ever before.

Adaptive test techniques have been used in industry for many years now. Applications of machine learning (ML) techniques have also been demonstrated and widely used in test today. This has opened opportunities to really push the envelope in terms of improving outgoing quality and while at the same time reducing production test costs. To take advantage of what ML has to offer, a key is to make sure our test infrastructure such as test hardware, software has a tight integration with ML techniques, so we can take advantage of both, test infrastructure optimizations and progress in ML techniques to deliver the best result for the business. When designing our ML algorithms, we must think not only of its accuracy, efficiency etc., but also pay attention to systems where it will run and optimize them to be best in class. Similarly, when designing our test hardware and software for manufacturing, we must look at the entire stack: automation systems that handle manufacturing routes, to testers, handlers, to operating systems that run on testers and make sure they are optimized to consume the output of ML algorithms and can move the units accordingly, to give the best throughput. Finally, the test engineers must also think about what test data they generate, what format is it in, how is it stored and is it optimized for absorption by ML applications so that they can take advantage of the data without human interaction.

This article will discuss how test world is ideally suited to apply ML techniques, what possibilities exist for improving our test techniques and test manufacturing (MFG) flows and therefore opportunities for future development.

1. Introduction:

Today, machine learning (ML) and artificial intelligence (AI) are the buzz words, we see many applications of these in almost every field, from faster and more optimized web search engines, to predicting users' shopping preferences, movie preferences to predicting outcomes of elections to medical field for diagnosis etc. and many more. In many of these applications, human behaviors and choices that are inherently unpredictable (example: what movie do I want to watch next) do impact the outcomes. Field of Si testing on the other hand, varies significantly. The Si devices that we see in our everyday lives, like the ones in our laptops, start with a design. It is then fabricated using a specific fabrication technology (or node) and after that, it goes through stress steps to weed out early life failures. After these 3 steps, the Si device characteristics, for the most part, are set – it doesn't have mind of its own to start acting differently for the same input stimuli (Figure 1). This makes application of ML to predict the behavior of these parts during TEST the most apt use of ML and AI, provided we have the right data (input features) and have the right infrastructure in test environment to take advantage of power of ML.

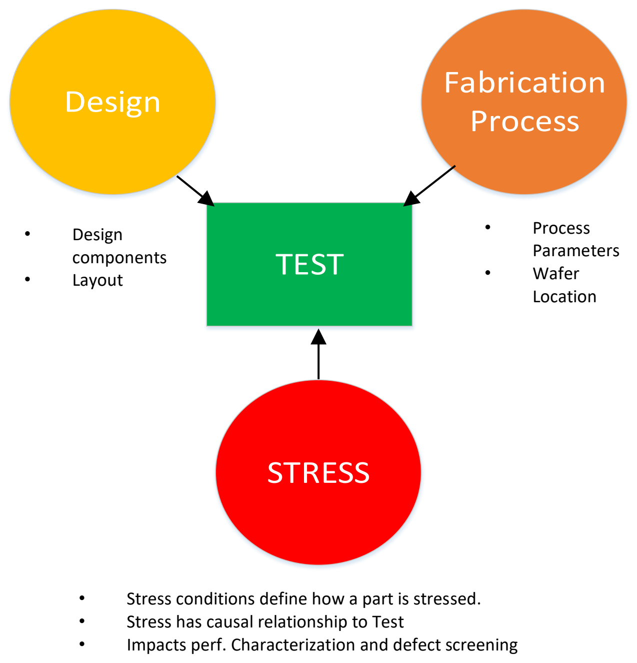

Design
Fabrication Process
TEST
STRESS
• Design components
• Layout
• Process Parameters
• Wafer Location
• Stress conditions define how a part is stressed.
• Stress has causal relationship to Test
• Impacts perf. Characterization and defect screening

Figure 1: Factors that impact TEST.

In HVM, test engineers have access to data from every step of test manufacturing process, like wafer sort, class test, system test etc. These large volumes of data sets can be used to solve improvement challenges along many different vectors such as yield, quality, performance, or test time improvements using ML techniques that are ubiquitously available (eg: xgboost, random forest etc.) in various ML packages today. These techniques can be used to show, using data analysis offline, what we can gain in any of the vectors listed above. For example, we can do an offline analysis of input features we have to show what test plans are not needed for specific units, or which types of units are most likely to fail in our manufacturing flows. Whether we can take advantage of these techniques to improve our manufacturing, really comes down to the test infrastructure in place and do we have ML integrated within our manufacturing flows at every level, from lot movement to test selection for device under test (DUT) while it is socketed on a test equipment.

It is therefore important that to deliver the most cost-effective test solution, we should be paying close attention to generating the right type of data, the format of how it is stored and investing in making manufacturing systems that can take advantage of immense value ML solutions can provide.

2. Architecting test manufacturing to leverage adaptive test techniques

Use of adaptive test techniques in test manufacturing is quite prevalent in industry and has been documented extensively in conference and journal papers. What we will discuss here is how to architect the manufacturing flows to leverage the best out of ML towards delivering the most cost effective and quality conscious output to our businesses.

A typical manufacturing flow for backend test consists of many steps as shown below in Figure 2.

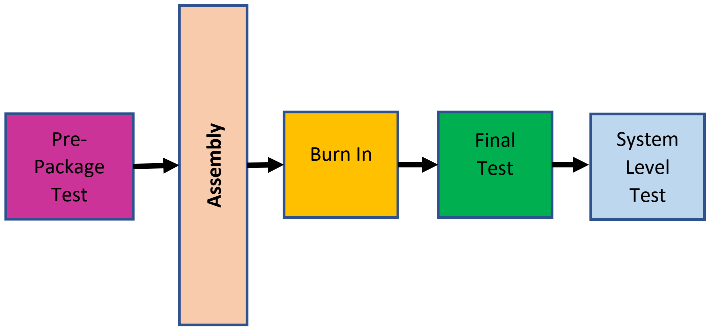


Figure 2: High level backend TEST MFG flow.

ML techniques can enhance the decisions on how we adapt our test plans to different units. For example: once we are done testing all die pre-assembly, we can use the pre-assembly data to grade the die along any number of vectors that are critical to specific business end use; for example: yield, speed, power etc. This grading system can then be used to select the best combination of die that should go on a package. Such a scheme will be particularly useful as we enter the world of chips coming from different vendors, all going on a single package to make the final device.

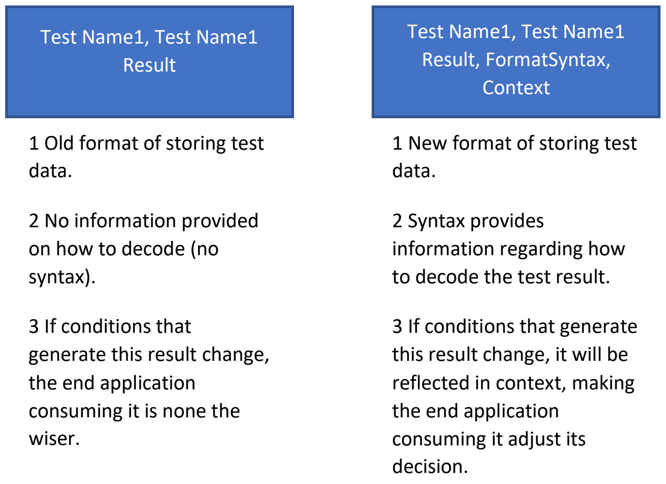
The other aspect is optimizing our test flows using ML. All the die that are printed on a wafer need not be tested the same way. As mentioned in previous paragraphs, we can learn from our data and with use of ML techniques, start creating the most optimized test flow for each die. Doing this analysis offline is great, but how can we deliver that to be effective in extracting business values as measured by test time, yields, run rate, outgoing quality? To do this, we must architect the entire manufacturing system in such a way that we have different adaptive test levers: all the way from updating test lot's route based on ML predictions to all the way down to the most granular level of selecting which test patterns run on which units in which socket(s). Such a system gives us the most control over unit movement and content delivery. Such a system also requires that our test hardware and software systems be able to house our ML engines in the flow and able to interact with it – giving it data and taking directions from it. To do this, all manufacturing systems: from those that control lot movement to the operating systems that run on testers, need to have some basic requirements: like ability to run industry standard ML packages, a handshake protocol with data bases on querying for information and writing to them.

3. Sources of data to consider for optimizing key MFG indicators

Key manufacturing performance indicators (KPIs) that have significant business impact are typically outgoing quality, yields, performance bucketing, test time in each socket, overall throughput, or cycle time of test factories. When we think about designing adaptive test MFG flows with ML to optimize these indicators, we as test engineers have to think about the sources of data that we can use towards this, what are ingredients of a complete data, what format should we be writing it in and what type of tests should we be writing to help us along the way. This last topic we will discuss in next section.

As far as sources of data goes, in general, we want to tap into as many sources that will assist us in making quality ML models towards improving our KPIs. This will include fabrication process data, design rule data, test data as the die/unit makes its way through manufacturing flow and data from test equipments' used that give us information about the tool health. Here, we will specifically focus on test data.

In a very simplistic form, test data consists of name of the test that is being performed on the DUT and its output, with the output giving us information about what happened during the test. We can also include more granular pieces of information to know details on exact events that happened during the test. For any given test, the actual syntax of what is written as test output will depend on how the test engineer wishes to see the results, tester operating system, how the output is connected to the data bases that store the data. In such a system, you need the test engineer to interpret the data for you. See left hand side, Figure 3. As we start working on products where we expect to get different chips from different vendors, such a system will not scale.



Test Name1, Test Name1 Result	Test Name1, Test Name1 Result, FormatSyntax, Context
1 Old format of storing test data.	1 New format of storing test data.
2 No information provided on how to decode (no syntax).	2 Syntax provides information regarding how to decode the test result.
3 If conditions that generate this result change, the end application consuming it is none the wiser.	3 If conditions that generate this result change, it will be reflected in context, making the end application consuming it adjust its decision.

Figure 3: Example of context aware test data (on right).

We need the output of our testing to have enough information so that it is consumable by other applications, like a ML engine, without the aid of the test engineer. To do this, the test data should have 3 components: test result + its context + test result format syntax (aka instructions on how to consume it). This is shown in right hand side of Figure 3. Here, test result is the raw result that comes out of running the test. Context will have the information of what were the conditions that defined the test instances (voltage, frequency, temperature, any other parameter that influences the output of the test). The third part, written here as test result format syntax, is a set of instructions on how to consume a given test result. One test engineer might for example write 40 different values indicating a specific measurement on each of the sub IPs. The instructions will give that information to end user.

The significance of the two fields here, outside of actual raw test result is quite enormous. It allows for a machine-readable format, with rules on how to understand it without having to sit down with a test engineer. But more importantly, by adding context to the data we generate, we make sure the ML models get applied only when the test results are generated with the right context. Any change in context will be identified and our output will adjust to the new context. This is an area where we need to have industry standard tools, although adoption of such tools will not be easy.

4. Prediction Measurements (PM):

Each test socket may potentially generate vast amounts of data. We tend to use what is generated and apply machine learning techniques to see if we can extract value towards improving one or more of our key indicators. However, a more deliberate approach would be to insert observations (tests) during our manufacturing flow that are not necessarily geared towards testing the device to its spec, but more towards giving us more information about the performance of individual electrical devices, like transistors, diodes, circuits etc. that can help us predict the performance of the complex logic or application blocks they are part of in the final CPU or chip. This is what we call prediction measurements or PMs. As an example: we know that the analog measurements that come from basic structures like a PLL or a digital sensor or an oscillator tend to be a good predictor of performance of the digital logic tests. We know today as industry is chasing higher and higher interface speeds with say PCIE etc., it is becoming harder and more expensive to test these IPs as part of our test flows. Can we then add specific readouts in our test flows that can be used to predict say speed performance of these IPs? Can we extend that logic to also create data sources that can predict our system performance for not only these IPs, but any other feature in general? We need more research done in this area from both industry and academia.

5. A glimpse of future:

This article will not be complete without thinking about what we as a test community should strive to do. As mentioned throughout this article, test is the most apt field for applications of ML. How much we can leverage will really come down to: are we writing the right types of tests, are we storing the right types of data and its context and in right formats, are we designing our manufacturing test infrastructure that makes it easy for ML engines to be tightly integrated within our test flows so we can control the movement of the units and content delivery. After we make all of this happen, we still need the ability to create ML models that are going to give us the right solutions towards optimizing our KPIs. Towards this, we must invest in developing systems that will make the domain experts in test state the constraints of the problem they want the ML solution to optimize against, specify the data sets and out comes the model. This is what happens typically when a test engineer sits down with a data scientist and for pushing the envelope of ML applications, we still absolutely need this interaction, no doubt about that. However, for many day-to-day use cases, we can empower the test engineers, who are domain experts in test, but may not have requisite training in the art of machine learning. This, sometimes, can create a barrier to openly embracing and applying ML techniques to problems at hand. Creating ML as a service (MLAAS), that can be called by anyone, anywhere and that can be given problem constraints and data sets, to create an output, is where there is scope for industry and academia research. Again, adopting these techniques across industry can be a challenge, but this will create a host of ideas and solutions that will push this subject matter forward.

Finally, with the aid of ML and test infrastructure innovations, our goal is to build a self-monitoring, self-correcting MFG flow, where our ML engines can monitor the data generated within the systems and react to that data real time, and our test infrastructure solutions can take the signals from our ML engine and correct our flows where necessary towards the most optimized KPIs that our businesses ask for.